Intel® Mobile Celeron™ Processor Specification Update

Release Date: March 1999

Order Number: 244444-002

The Intel® Mobile Celeron™ processor or the Intel® Celeron™ Processor Mobile Module may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are documented in this Specification Update.

CONTENTS

REVISION HISTORY

Date of Revision	Version	Description
January 1999	001	This document is the first Specification Update for the Intel® Mobile Celeron™ processor.
March 1999	002	Updated the Documentation Changes and Specifications Clarifications sections.

PREFACE

This document is an update to the specifications contained in the *Intel® Mobile Celeron™ Processor (BGA) Datasheet* (Order Number 245106-001), the *Intel® Mobile Celeron Processor in Mobile Module MMC-1* (Order Number 245101-001), the *Intel® Mobile Celeron Processor in Mobile Module MMC-2* (Order Number 245102-001) and the *Intel Architecture Software Developer's Manual*, *Volumes 1, 2 and 3* (Order Numbers 243190, 243191, and 243192, respectively). It is intended for hardware system manufacturers and software developers of applications, operating systems, or tools. It contains Specification Changes, S-Specs, Errata, Specification Clarifications, and Documentation Changes.

Nomenclature

S-Spec Number is a five digit code used to identify products. Products are differentiated by their unique characteristics, e.g., core speed, L2 cache size, package type, etc. as described in the processor identification information table. Care should be taken to read all notes associated with each S-Spec number.

Specification Changes are modifications to the current published specifications for the Intel® Mobile Celeron™ processor or the Intel® Celeron™ Processor Mobile Module. These changes will be incorporated in the next release of the specifications.

Specification Clarifications describe a specification in greater detail or further highlight a specification's impact to a complex design situation. These clarifications will be incorporated in the next release of the specifications.

Documentation Changes include typos, errors, or omissions from the current published specifications. These changes will be incorporated in the next release of the specifications.

Errata are design defects or errors. Errata may cause the Intel Mobile Celeron processor or the Intel Mobile Celeron Module's behavior to deviate from published specifications. Hardware and software designed to be used with any given processor must assume that all errata documented for that processor are present on all devices unless otherwise noted.

Identification Information

The Intel Mobile Celeron processor or the Intel Celeron Processor Mobile Module can be identified by the following values:

Family[1]	**266- , 300- Model 6[2]**
0110	0110

NOTES:

1. The Family corresponds to bits [11:8] of the EDX register after RESET, bits [11:8] of the EAX register after the CPUID instruction is executed with a 1 in the EAX register, and the generation field of the Device ID register accessible through Boundary Scan.
2. The Model corresponds to bits [7:4] of the EDX register after RESET, bits [7:4] of the EAX register after the CPUID instruction is executed with a 1 in the EAX register, and the model field of the Device ID register accessible through Boundary Scan.

The Intel Mobile Celeron processor and the Intel Celeron Processor Mobile Module's second level (L2) cache size can be determined by the following register contents:

128-Kbyte Unified L2 Cache[1]	41h

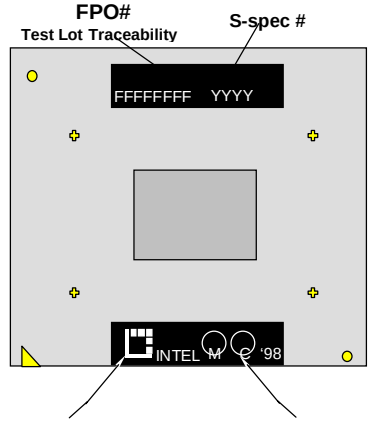
NOTE:

1. For the Intel® Mobile Celeron™ processor and the Intel® Celeron™ Processor Mobile Module, the unified L2 cache size will be returned as one of the cache/TLB descriptors when the CPUID instruction is executed with a 2 in the EAX register.

Specification Update for Intel® Mobile Celeron™ Processors

GENERAL INFORMATION

Intel® Mobile Celeron™ Processor (BGA) Markings



- Supplier Lot ID
- Serial Number

Intel® Celeron™ Processor Mobile Module Markings

The Product Tracking Code (PTC) determines the Intel assembly level of the module. The PTC is on the secondary side of the module and provides the following information:

Example: ***PMG33302001AA***

- The PTC will consist of 13 characters as identified in the above example and can be broken down as follows:

AABCCCDDEEEFF

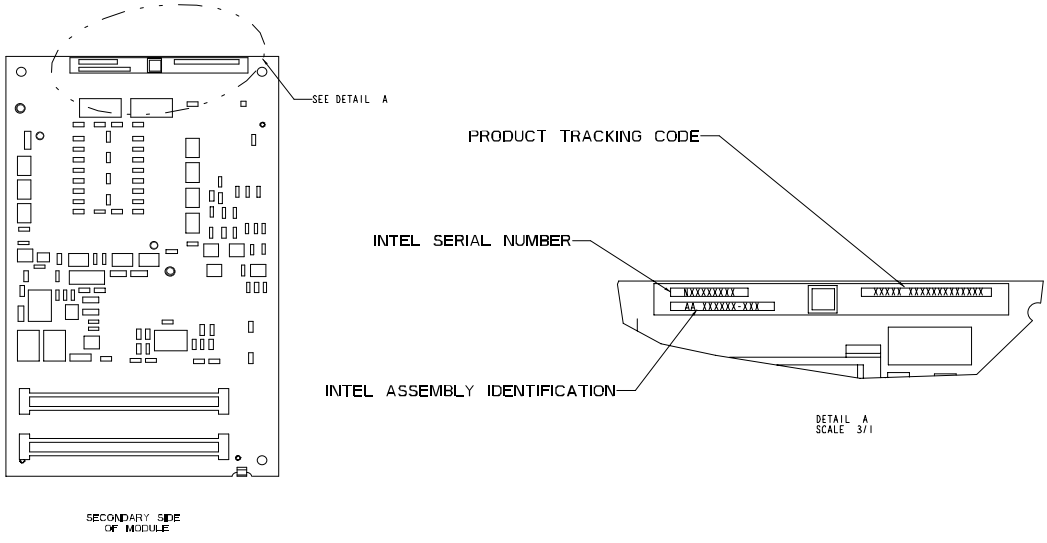
- Definition:

AA	-	Processor Module = PM
B	-	Intel® Celeron™ Processor Mobile Module = H
CCC	-	Speed Identity = 300 and 266
DD	-	Cache Size = 01 (128KB)
EEE	-	Notifiable Design Revision (Start at 001)
FF	-	Notifiable Processor Revision (Start at AA)

NOTE:

For other Intel Mobile Modules, the second field (B) is defined as:

- Pentium® II Processor Mobile Module (MMC-1) at 266/300 MHz = D
- Pentium II Processor Mobile Module (MMC-2) at 266/300 MHz = E
- Pentium II Processor Mobile Module (MMC-1) at 266PE/300PE/333/366 MHz = F
- Pentium II Processor Mobile Module (MMC-2) at 266PE/300PE/333/366 MHz = G
- Intel® Celeron™ Processor Mobile Module (MMC-2) = H
- Intel Celeron Processor Mobile Module (MMC-2) = I



Intel® Celeron™ Processor Mobile Module (MMC-1)

Intel® Mobile Celeron™ Processor Identification Information

S-Spec	Product Stepping	CPUID	Speed (MHz) Core/Bus	Integrated L2 Size (Kbytes)	Package	Notes
SL23Y	mcbA0	066Ah	266/66	128	BGA1	1
SL3AH	mcbA0	066Ah	300/66	128	BGA1	1

NOTE:

1. VCC_CORE is specified for 1.6 V +/-135mV for these Intel® Mobile Celeron™ processors (BGA1 package).

Intel® Celeron™ Processor Mobile Module Identification Information

PTC	Product Stepping	CPUID	Speed (MHz) Core/Bus	Integrated L2 Size (Kbytes)	Package	Notes
PMH26601001AA	cmmA0	066Ah	266/66	128	MMC1	1, 2
PMH30001001AA	cmmA0	066Ah	300/66	128	MMC1	1, 2
PMI30001001AA	cmmA0	066Ah	300/66	128	MMC2	1, 2

NOTES:

1. Vcore voltage is 1.6 V.
2. Voltage regulator comparator modification.

Summary Table of Changes

The following table indicates the Specification Changes, Errata, Specification Clarifications, or Documentation Changes which apply to the Intel Mobile Celeron processors. Intel intends to fix some of the errata in a future stepping of the component, and to account for the other outstanding issues through documentation or specification changes as noted. This table uses the following notations:

CODES USED IN SUMMARY TABLE

X:	Specification Change, Erratum, Specification Clarification, or Documentation Change applies to the given processor stepping.
Doc:	Intel intends to update the appropriate documentation in a future revision.
Fix:	This erratum is intended to be fixed in a future stepping of the component.
Fixed:	This erratum has been previously fixed.
NoFix:	There are no plans to fix this erratum.
(No mark) or (blank box):	This item is fixed in or does not apply to the given stepping.
AP:	APIC related erratum.
SUB:	This column refers to errata on the Intel® Mobile Celeron™ processor or the Intel® Celeron™ Processor Mobile Module substrate.
Shaded:	This item is either new or modified from the previous version of the document.

Some of Intel's Specification Updates will be undergoing a numbering methodology change to reduce confusion when referring to errata which affect a specific product. Each Specification Update item will be prefixed with a capital letter to distinguish the product it refers to. The key below details the letters which will be used for the current Intel microprocessor Specification Updates:

A = Pentium® II processor

B = Mobile Pentium II processor

C = Intel® Celeron™ processor

D = Pentium® II Xeon™ processor

E = Pentium® III processor

G = Pentium® III Xeon™ processor

H = Intel® Mobile Celeron™ processor

The Specification Updates for the Pentium processor, Pentium Pro processor, and other Intel products will not be implementing such a convention at this time.

NO.	mcbA0	cmmA0	SUB	Plans	ERRATA
H1	X	X		NoFix	FP Data Operand Pointer may be incorrectly calculated after FP access which wraps 64-Kbyte boundary in 16-bit code
H2	X	X		NoFix	Differences exist in debug exception reporting
H3	X	X		NoFix	Code fetch matching disabled debug register may cause debug exception
H4	X	X		NoFix	Double ECC error on read may result in BINIT#
H5	X	X		NoFix	FP inexact-result exception flag may not be set
H6	X	X		NoFix	BTM for SMI will contain incorrect FROM EIP
H7	X	X		NoFix	I/O restart in SMM may fail after simultaneous MCE
H8	X	X		NoFix	Branch traps do not function if BTMs are also enabled
H9	X	X		NoFix	Machine check exception handler may not always execute successfully
H10	X	X		NoFix	MCE due to L2 parity error gives L1 MCACOD.LL
H11	X	X		NoFix	LBER may be corrupted after some events
H12	X	X		NoFix	BTMs may be corrupted during simultaneous L1 cache line replacement
H13	X	X		Fix	Potential early deassertion of LOCK# during split-lock cycles
H14	X	X		NoFix	A20M# may be inverted after returning from SMM and Reset
H15	X	X		Fix	Reporting of floating-point exception may be delayed
H16	X	X		NoFix	Near CALL to ESP creates unexpected EIP address
H17	X	X		Fix	Built-in self test always gives nonzero result
H18	X	X		Fix	Cache state corruption in the presence of page A/D-bit setting and snoop traffic
H19	X	X		Fix	Snoop cycle generates spurious machine check exception
H20	X	X		Fix	MOVD/MOVQ instruction writes to memory prematurely
H21	X	X		NoFix	Memory type undefined for nonmemory operations
H22	X	X		NoFix	FP Data Operand Pointer may not be zero after power on or Reset
H23	X	X		NoFix	MOVD following zeroing instruction can cause incorrect result
H24	X	X		NoFix	Premature execution of a load operation prior to exception handler invocation
H25	X	X		NoFix	Read portion of RMW instruction may execute twice
H26	X	X		Fix	Intervening writeback may occur during locked transaction
H27	X	X		NoFix	MC2_STATUS MSR has model-specific error code and machine check architecture error code reversed
H28	X	X		NoFix	Mixed cacheability of lock variables is problematic in MP systems
H29		X		Fix	Thermal sensor may assert SMBALERT# incorrectly

NO.	mcbA0	cmmA0	SUB	Plans	ERRATA
H30	X	X		NoFix	MOV with debug register causes debug exception
H31	X	X		NoFix	Upper four PAT entries not usable with Mode B or Mode C paging
H32	X	X		Fix	Incorrect memory type may be used when MTRRs are disabled
H33	X	X		Fix	Misprediction in program flow may cause unexpected instruction execution
H34	X	X		NoFix	Data breakpoint exception in a displacement relative near call may corrupt EIP
H35	X	X		NoFix	System bus ECC not functional with 2:1 ratio
H36	X	X		Fix	Fault on REP CMPS/SCAS operation may cause incorrect EIP
H37	X	X		NoFix	RDMSR and WRMSR to invalid MSR address may not cause GP fault
H38	X	X		NoFix	SYSENTER/SYSEXIT instructions can implicitly load “null segment selector” to SS and CS registers
H39	X	X		NoFix	PRELOAD followed by EXTEST does not load boundary scan data
H40	X	X		NoFix	Far jump to new TSS with D-bit cleared may cause system hang
H41	X	X		NoFix	Incorrect chunk ordering may prevent execution of the Machine Check Exception handler after BINIT#
H42	X	X		NoFix	Resume Flag may not be cleared after debug exception

NO.	mcbA0	cmmA0	SUB	Plans	DOCUMENTATION CHANGES
H1	X	X		Doc	SMBus data setup time

ERRATA

H1. FP Data Operand Pointer May Be Incorrectly Calculated After FP Access Which Wraps 64-Kbyte Boundary in 16-Bit Code

PROBLEM: The FP Data Operand Pointer is the effective address of the operand associated with the last noncontrol floating-point instruction executed by the machine. If an 80-bit floating-point access (load or store) occurs in a 16-bit mode other than protected mode (in which case the access will produce a segment limit violation), the memory access wraps a 64-Kbyte boundary, and the floating-point environment is subsequently saved, the value contained in the FP Data Operand Pointer may be incorrect.

IMPLICATION: A 32-bit operating system running 16-bit floating-point code may encounter this erratum, under the following conditions:

- The operating system is using a segment greater than 64 Kbytes in size.
- An application is running in a 16-bit mode other than protected mode.
- An 80-bit floating-point load or store which wraps the 64-Kbyte boundary is executed.
- The operating system performs a floating-point environment store (FSAVE/FNSAVE/FSTENV/FNSTENV) after the above memory access.
- The operating system uses the value contained in the FP Data Operand Pointer.

Wrapping an 80-bit floating-point load around a segment boundary in this way is not a normal programming practice. Intel has not currently identified any software which exhibits this behavior.

WORKAROUND: If the FP Data Operand Pointer is used in an OS which may run 16-bit floating-point code, care must be taken to ensure that no 80-bit floating-point accesses are wrapped around a 64-Kbyte boundary.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

H2. Differences Exist in Debug Exception Reporting

PROBLEM: There exist some differences in the reporting of code and data breakpoint matches between that specified by previous Intel processors' specifications and the behavior of the Intel Mobile Celeron processor, as described below:

CASE 1:

The first case is for a breakpoint set on a MOVSS or POPSS instruction, when the instruction following it causes a debug register protection fault (DR7.gd is already set, enabling the fault). The processor reports delayed data breakpoint matches from the MOVSS or POPSS instructions by setting the matching DR6.bi bits, along with the debug register protection fault (DR6.bd). If additional breakpoint faults are matched during the call of the debug fault handler, the processor sets the breakpoint match bits (DR6.bi) to reflect the breakpoints matched by both the MOVSS or POPSS breakpoint and the debug fault handler call. The Intel Mobile Celeron processor only sets DR6.bd in either situation, and does not set any of the DR6.bi bits.

CASE 2:

In the second breakpoint reporting failure case, if a MOVSS or POPSS instruction with a data breakpoint is followed by a store to memory which crosses a 4- Kbyte page boundary, the breakpoint information for the MOVSS or POPSS will be lost. Previous processors retain this information across such a page split.

CASE 3:

If they occur after a MOVSS or POPSS instruction, the INT *n*, INTO, and INT3 instructions zero the DR6.bi bits (bits B0 through B3), clearing pending breakpoint information, unlike previous processors.

CASE 4:

If a data breakpoint and an SMI (System Management Interrupt) occur simultaneously, the SMI will be serviced via a call to the SMM handler, and the pending breakpoint will be lost.

CASE 5:

When an instruction which accesses a debug register is executed, and a breakpoint is encountered on the instruction, the breakpoint is reported twice.

IMPLICATION: When debugging or when developing debuggers for a Intel Mobile Celeron processor-based system, this behavior should be noted. Normal usage of the MOVSS or POPSS instructions (e.g., following them with a MOV ESP) will not exhibit the behavior of cases 1-3. Debugging in conjunction with SMM will be limited by case 4.

WORKAROUND: Following MOVSS and POPSS instructions with a MOV ESP instruction when using breakpoints will avoid the first three cases of this erratum. No workaround has been identified for cases 4 or 5.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

H3. Code Fetch Matching Disabled Debug Register May Cause Debug Exception

PROBLEM: The bits L0-3 and G0-3 enable breakpoints local to a task and global to all tasks, respectively. If one of these bits is set, a breakpoint is enabled, corresponding to the addresses in the debug registers DR0 - DR3. If at least one of these breakpoints is enabled, any of these registers are *disabled* (e.g., L*n* and G*n* are 0), and RW*n* for the disabled register is 00 (indicating a breakpoint on instruction execution), normally an instruction fetch will not cause an instruction-breakpoint fault based on a match with the address in the disabled register(s). However, if the address in a disabled register matches the address of a code fetch which also results in a page fault, an instruction-breakpoint fault will occur.

IMPLICATION: While debugging software, extraneous instruction-breakpoint faults may be encountered if breakpoint registers are not cleared when they are disabled. Debug software which does not implement a code breakpoint handler will fail, if this occurs. If a handler is present, the fault will be serviced. Mixing data and code may exacerbate this problem by allowing disabled data breakpoint registers to break on an instruction fetch.

WORKAROUND: The debug handler should clear breakpoint registers before they become disabled.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

H4. Double ECC Error on Read May Result in BINIT#

PROBLEM: For this erratum to occur, the following conditions must be met:

- Machine Check Exceptions (MCEs) must be enabled.
- A dataless transaction (such as a write invalidate) must be occurring simultaneously with a transaction which returns data (a normal read).
- The read data must contain a double-bit uncorrectable ECC error.

If these conditions are met, the Intel Mobile Celeron processor will not be able to determine which transaction was erroneous, and instead of generating an MCE, it will generate a BINIT#.

IMPLICATION: The bus will be reinitialized in this case. However, since a double-bit uncorrectable ECC error occurred on the read, the MCE handler (which is normally reached on a double-bit uncorrectable ECC error for a read) would most likely cause the same BINIT# event.

WORKAROUND: Though the ability to drive BINIT# can be disabled in the Intel Mobile Celeron processor, which would prevent the effects of this erratum, overall system behavior would not improve, since the error which would normally cause a BINIT# would instead cause the machine to shut down. No other workaround has been identified.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

H5. FP Inexact-Result Exception Flag May Not Be Set

PROBLEM: When the result of a floating-point operation is not exactly representable in the destination format (1/3 in binary form, for example), an inexact-result (precision) exception occurs. When this occurs, the PE bit (bit 5 of the FPU status word) is normally set by the processor. Under certain rare conditions, this bit may not be set when this rounding occurs. However, other actions taken by the processor (invoking the software exception handler if the exception is unmasked) are not affected. This erratum can only occur if the floating-point operation which causes the precision exception is immediately followed by one of the following instructions:

- FST m32real
- FST m64real
- FSTP m32real
- FSTP m64real
- FSTP m80real
- FIST m16int
- FIST m32int
- FISTP m16int
- FISTP m32int
- FISTP m64int

Note that even if this combination of instructions is encountered, there is also a dependency on the internal pipelining and execution state of both instructions in the processor.

IMPLICATION: Inexact-result exceptions are commonly masked or ignored by applications, as it happens frequently, and produces a rounded result acceptable to most applications. The PE bit of the FPU status word may not always be set upon receiving an inexact-result exception. Thus, if these exceptions are unmasked, a floating-point error exception handler may not recognize that a precision exception occurred. Note that this is a “sticky” bit, e.g., once set by an inexact-result condition, it remains set until cleared by software.

WORKAROUND: This condition can be avoided by inserting two NOP instructions between the two floating-point instructions.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

H6. BTM for SMI Will Contain Incorrect FROM EIP

PROBLEM: A system management interrupt (SMI) will produce a Branch Trace Message (BTM), if BTMs are enabled. However, the FROM EIP field of the BTM (used to determine the address of the instruction which was being executed when the SMI was serviced) will not have been updated for the SMI, so the field will report the same FROM EIP as the previous BTM.

IMPLICATION: A BTM which is issued for an SMI will not contain the correct FROM EIP, limiting the usefulness of BTMs for debugging software in conjunction with System Management Mode (SMM).

WORKAROUND: None identified at this time.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

H7. I/O Restart in SMM May Fail After Simultaneous MCE

PROBLEM: If an I/O instruction (IN, INS, REP INS, OUT, OUTS, or REP OUTS) is being executed, and if the data for this instruction becomes corrupted, the Intel Mobile Celeron processor will signal a machine check exception (MCE). If the instruction is directed at a device which is powered down, the processor may also receive an assertion of SMI#. Since MCEs have higher priority, the processor will call the MCE handler, and the SMI# assertion will remain pending. However, upon attempting to execute the first instruction of the MCE handler, the SMI# will be recognized and the processor will attempt to execute the SMM handler. If the SMM handler is completed successfully, it will attempt to restart the I/O instruction, but will not have the correct machine state, due to the call to the MCE handler.

IMPLICATION: A simultaneous MCE and SMI# assertion may occur for one of the I/O instructions above. The SMM handler may attempt to restart such an I/O instruction, but will have corrupted state due to the MCE handler call, leading to failure of the restart and shutdown of the processor.

WORKAROUND: If a system implementation must support both SMM and MCEs, the first thing the SMM handler code (when an I/O restart is to be performed) should do is check for a pending MCE. If there is an MCE pending, the SMM handler should immediately exit via an RSM instruction and allow the machine check exception handler to execute. If there is not, the SMM handler may proceed with its normal operation.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

H8. Branch Traps Do Not Function if BTMs Are Also Enabled

PROBLEM: If branch traps or branch trace messages (BTMs) are enabled alone, both function as expected. However, if both are enabled, only the BTMs will function, and the branch traps will be ignored.

IMPLICATION: The branch traps and branch trace message debugging features cannot be used together.

WORKAROUND: If branch trap functionality is desired, BTMs must be disabled.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

H9. Machine Check Exception Handler May Not Always Execute Successfully

PROBLEM: An asynchronous machine check exception (MCE), such as a BINIT# event, which occurs during an access that splits a 4-Kbyte page boundary may leave some internal registers in an indeterminate state. Thus, MCE handler code may not always run successfully if an asynchronous MCE has occurred previously.

IMPLICATION: An MCE may not always result in the successful execution of the MCE handler. However, asynchronous MCEs usually occur upon detection of a catastrophic system condition that would also hang the processor. Leaving MCEs disabled will result in the condition which caused the asynchronous MCE instead causing the processor to enter shutdown. Therefore, leaving MCEs disabled may not improve overall system behavior.

WORKAROUND: No workaround which would guarantee successful MCE handler execution under this condition has been identified.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

H10. MCE Due to L2 Parity Error Gives L1 MCACOD.LL

PROBLEM: If a Cache Reply Parity (CRP) error, Cache Address Parity (CAP) error, or Cache Synchronous Error (CSER) occurs on an access to the Intel Mobile Celeron processor's L2 cache, the resulting Machine Check Architectural Error Code (MCACOD) will be logged with '01' in the LL field. This value indicates an L1 cache error; the value should be '10', indicating an L2 cache error. Note that L2 ECC errors have the correct value of '10' logged.

IMPLICATION: An L2 cache access error, other than an ECC error, will be improperly logged as an L1 cache error in MCACOD.LL.

WORKAROUND: None identified at this time.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

H11. LBER May Be Corrupted After Some Events

PROBLEM: The last branch record (LBR) and the last branch before exception record (LBER) can be used to determine the source and destination information for previous branches or exceptions. The LBR contains the source and destination addresses for the last branch or exception, and the LBER contains similar information for the last branch taken before the last exception. This information is typically used to determine the location of a branch which leads to execution of code which causes an exception. However, after a catastrophic bus condition which results in an assertion of BINIT# and the reinitialization of the buses, the value in the LBER may be corrupted. Also, after either a CALL which results in a fault or a software interrupt, the LBER and LBR will be updated to the same value, when the LBER should not have been updated.

IMPLICATION: The LBER and LBR registers are used only for debugging purposes. When this erratum occurs, the LBER will not contain reliable address information. The value of LBER should be used with caution when debugging branching code; if the values in the LBR and LBER are the same, then the LBER value is incorrect. Also, the value in the LBER should not be relied upon after a BINIT# event.

WORKAROUND: None identified at this time.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

H12. BTMs May Be Corrupted During Simultaneous L1 Cache Line Replacement

PROBLEM: When Branch Trace Messages (BTMs) are enabled and such a message is generated, the BTM may be corrupted when issued to the bus by the L1 cache if a new line of data is brought into the L1 data cache simultaneously. Though the new line being stored in the L1 cache is stored correctly, and no corruption occurs in the data, the information in the BTM may be incorrect due to the internal collision of the data line and the BTM.

IMPLICATION: Although BTMs may not be entirely reliable due to this erratum, the conditions necessary for this boundary condition to occur have only been exhibited during focused simulation testing. Intel has currently not observed this erratum in a system level validation environment.

WORKAROUND: None identified at this time.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

H13. Potential Early Deassertion of LOCK# During Split-Lock Cycles

PROBLEM: During a split-lock cycle there are four bus transactions: 1st ADS# (a partial read), 2nd ADS# (a partial read), 3rd ADS# (a partial write), and the 4th ADS# (a partial write). Due to this erratum, LOCK# may deassert one clock after the 4th ADS# of the split-lock cycle instead of after the 4th RS# assertion corresponding to the 4th ADS# has been sampled. The following sequence of events are required for this erratum to occur:

1. A lock cycle occurs (split or nonsplit).
2. Five more bus transactions (assertion of ADS#) occur.
3. A split-lock cycle occurs and BNR# toggles after the 3rd ADS# (partial write) of the split-lock cycle. This in turn delays the assertion of the 4th ADS# of the split-lock cycle. BNR# toggling at this time could most likely happen when the bus is set for an IOQ depth of 2.

When all of these events occur, LOCK# will be deasserted in the next clock after the 4th ADS# of the split-lock cycle.

IMPLICATION: This may affect chipset logic which monitors the behavior of LOCK# deassertion.

WORKAROUND: None identified at this time.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

H14. A20M# May Be Inverted After Returning From SMM and Reset

PROBLEM: This erratum is seen when software causes the following events to occur:

1. The assertion of A20M# in real address mode.
2. After entering the 1-Mbyte address wrap-around mode caused by the assertion of A20M#, there is an assertion of SMI# intended to cause a Reset or remove power to the processor. Once in the SMM handler, software saves the SMM state save map to an area of nonvolatile memory from which it can be restored at some point in the future. Then software asserts RESET# or removes power to the processor.
3. After exiting Reset or completion of power-on, software asserts SMI# again. Once in the SMM handler, it then retrieves the old SMM state save map which was saved in event 2 above and copies it into the current SMM state save map. Software then asserts A20M# and executes the RSM instruction. After exiting the SMM handler, the polarity of A20M# is inverted.

IMPLICATION: If this erratum occurs, A20M# will behave with a polarity opposite from what is expected (e.g., the 1-Mbyte address wrap-around mode is enabled when A20M# is deasserted, and does not occur when A20M# is asserted).

WORKAROUND: Software should save the A20M# signal state in nonvolatile memory before an assertion of RESET# or a power down condition. After coming out of Reset or at power on, SMI# should be asserted again. During the restoration of the old SMM state save map described in event 3 above, the entire map should be restored, except for bit 5 of the byte at offset 7F18h. This bit should retain the value assigned to it when the SMM

state save map was created in event 3. The SMM handler should then restore the original value of the A20M# signal.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

H15. Reporting of Floating-Point Exception May Be Delayed

PROBLEM: The Intel Mobile Celeron processor normally reports a floating-point exception for an instruction when the next floating-point or MMX™ technology instruction is executed. The assertion of FERR# and/or the INT 16 interrupt corresponding to the exception may be delayed until the floating-point or MMX technology instruction *after* the one which is expected to trigger the exception, if the following conditions are met:

1. A floating-point instruction causes an exception.
2. Before another floating-point or MMX™ technology instruction, any one of the following occurs:
   a. A subsequent data access occurs to a page which has not been marked as accessed, or
   b. Data is referenced which crosses a page boundary, or
   c. A possible page-fault condition is detected which, when resolved, completes without faulting.
3. The instruction causing event 2 above is followed by a MOVQ or MOVD store instruction.

IMPLICATION: This erratum only affects software which operates with floating-point exceptions unmasked. Software which requires floating-point exceptions to be visible on the next floating-point or MMX technology instruction, and which uses floating-point calculations on data which is then used for MMX technology instructions, may see a delay in the reporting of a floating-point instruction exception in some cases. Note that mixing floating-point and MMX technology instructions in this way is not recommended.

WORKAROUND: Inserting a WAIT or FWAIT instruction (or reading the floating-point status register) between the floating-point instruction and the MOVQ or MOVD instruction will give the expected results. This is already the recommended practice for software.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

H16. Near CALL to ESP Creates Unexpected EIP Address

PROBLEM: As documented, the CALL instruction saves procedure linking information in the procedure stack and jumps to the called procedure specified with the destination (target) operand. The target operand specifies the address of the first instruction in the called procedure. This operand can be an immediate value, a general purpose register, or a memory location. When accessing an absolute address indirectly using the stack pointer (ESP) as a base register, the base value used is the value in the ESP register before the instruction executes. However, when accessing an absolute address directly using ESP as the base register, the base value used is the value of ESP *after* the return value is pushed on the stack, not the value in the ESP register *before* the instruction executed.

IMPLICATION: Due to this erratum, the processor may transfer control to an unintended address. Results are unpredictable, depending on the particular application, and can range from no effect to the unexpected termination of the application due to an exception. Intel has observed this erratum only in a focused testing environment. Intel has not observed any commercially available operating system, application, or compiler that makes use of or generates this instruction.

WORKAROUND: If the other seven general purpose registers are unavailable for use, and it is necessary to do a CALL via the ESP register, first push ESP onto the stack, then perform an *indirect* call using ESP (e.g., CALL [ESP]). The saved version of ESP should be popped off the stack after the call returns.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

H17. Built-in Self Test Always Gives Nonzero Result

PROBLEM: The Built-in Self Test (BIST) of the Intel Mobile Celeron processor does not give a zero result to indicate a passing test. Regardless of pass or fail status, bit 6 of the BIST result in the EAX register after running BIST is set.

IMPLICATION: Software which relies on a zero result to indicate a passing BIST will indicate BIST failure.

WORKAROUND: Mask bit 6 of the BIST result register when analyzing BIST results.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

H18. Cache State Corruption in the Presence of Page A/D-bit Setting and Snoop Traffic

PROBLEM: If an operating system uses the Page Access and/or Dirty bit feature implemented in the Intel architecture and there is a significant amount of snoop traffic on the bus, while the processor is setting the Access and/or Dirty bit the processor may inappropriately change a single L1 cache line to the modified state.

IMPLICATION: The occurrence of this erratum may result in cache incoherency, which may cause parity errors, data corruption (with no parity error), unexpected application or operating system termination, or system hangs.

WORKAROUND: It is possible for BIOS code to contain a workaround for this erratum.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

H19. Snoop Cycle Generates Spurious Machine Check Exception

PROBLEM: The processor may incorrectly generate a Machine Check Exception (MCE) when it processes a snoop access that does not hit the L1 data cache. Due to an internal logic error, this type of snoop cycle may still check data parity on undriven data lines. The processor generates a spurious machine check exception as a result of this unnecessary parity check.

IMPLICATION: A spurious machine check exception may result in an unexpected system halt if Machine Check Exception reporting is enabled in the operating system.

WORKAROUND: It is possible for BIOS code to contain a workaround for this erratum. This workaround would fix the erratum; however, the data parity error will still be reported.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

H20. MOVD/MOVQ Instruction Writes to Memory Prematurely

PROBLEM: When an instruction encounters a fault, the faulting instruction should not modify any CPU or system state. However, when the MMX technology store instructions MOVD and MOVQ encounter any of the following events, it is possible for the store to be committed to memory even though it should be canceled:

1. If CR0.EM = 1 (Emulation bit), then the store could happen prior to the triggered invalid opcode exception.
2. If the floating-point Top-of-Stack (FP TOS) is not zero, then the store could happen prior to executing the processor assist routine that sets the FP TOS to zero.
3. If there is an unmasked floating-point exception pending, then the store could happen prior to the triggered unmasked floating-point exception.

4. If CR0.TS = 1 (Task Switched bit), then the store could happen prior to the triggered Device Not Available (DNA) exception.

· If the MOVD/MOVQ instruction is restarted after handling any of the above events, then the store will be performed again, overwriting with the expected data. The instruction will not be restarted after event 1. The instruction will definitely be restarted after events 2 and 4. The instruction may or may not be restarted after event 3, depending on the specific exception handler.

IMPLICATION: This erratum causes unpredictable behavior in an application if MOVD/MOVQ instructions are used to manipulate semaphores for multiprocessor synchronization, or if these MMX instructions are used to write to uncacheable memory or memory mapped I/O that has side effects, e.g., graphics devices. This erratum is completely transparent to all applications that do not have these characteristics. When each of the above conditions are analyzed:

1. Setting the CR0.EM bit forces all floating-point/MMX™ instructions to be handled by software emulation. The MOVD/MOVQ instruction, which is an MMX instruction, would be considered an invalid instruction. Operating systems typically terminates the application after getting the expected invalid opcode fault.
2. The FP TOS not equal to 0 case only occurs when the MOVD/MOVQ store is the first MMX instruction in an MMX technology routine and the previous floating-point routine did not clean up the floating-point states properly when it exited. Floating-point routines commonly leave TOS to 0 prior to exiting. For a store to be executed as the first MMX instruction in an MMX technology routine following a floating-point routine, the software would be implementing instruction level intermixing of floating-point and MMX instructions. Intel does not recommend this practice.
3. The unmasked floating-point exception case only occurs if the store is the first MMX technology instruction in an MMX technology routine and the previous floating-point routine exited with an unmasked floating-point exception pending. Again, for a store to be executed as the first MMX instruction in an MMX technology routine following a floating-point routine, the software would be implementing instruction level intermixing of floating-point and MMX instructions. Intel does not recommend this practice.
4. Device Not Available (DNA) exceptions occur naturally when a task switch is made between two tasks that use either floating-point instructions and/or MMX instructions. For this erratum, in the event of the DNA exception, data from the prior task may be temporarily stored to the present task's program state.

WORKAROUND: Do not use MMX instructions to manipulate semaphores for multiprocessor synchronization. Do not use MOVD/MOVQ instructions to write directly to I/O devices if doing so triggers user visible side effects. An OS can prevent old data from being stored to a new task's program state by cleansing the FPU explicitly after every task switch. Follow Intel's recommended programming paradigms in the *Intel Architecture Optimization Manual* for writing MMX technology programs. Specifically, do not mix floating-point and MMX instructions. When transitioning to new a MMX technology routine, begin with an instruction that does not depend on the prior state of either the MMX technology registers or the floating-point registers, such as a load or PXOR mm0, mm0. Be sure that the FP TOS is clear before using MMX instructions.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

H21. Memory Type Undefined for Nonmemory Operations

PROBLEM: The Memory Type field for nonmemory transactions such as I/O and Special Cycles are undefined. Although the Memory Type attribute for nonmemory operations logically should (and usually does) manifest itself as UC, this feature is not designed into the implementation and is therefore inconsistent.

IMPLICATION: Bus agents may decode a non-UC memory type for nonmemory bus transactions.

WORKAROUND: Bus agents must consider transaction type to determine the validity of the Memory Type field for a transaction.

STATUS: For the steppings affected, see the Summary Table of Changes at the beginning of this section.

H22. FP Data Operand Pointer May Not Be Zero After Power On or Reset

PROBLEM: The FP Data Operand Pointer, as specified, should be reset to zero upon power on or Reset by the processor. Due to this erratum, the FP Data Operand Pointer may be nonzero after power on or Reset.

IMPLICATION: Software which uses the FP Data Operand Pointer and count on its value being zero after power on or Reset without first executing an FINIT/FNINIT instruction will use an incorrect value, resulting on incorrect behavior of the software.

WORKAROUND: Software should follow the recommendation in Section 8.2 of the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide* (Order Number 243192). This recommendation states that if the FPU will be used, software-initialization code should execute an FINIT/FNINIT instruction following a hardware reset. This will correctly clear the FP Data Operand Pointer to zero.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

H23. MOVD Following Zeroing Instruction Can Cause Incorrect Result

PROBLEM: An incorrect result may be calculated after the following circumstances occur:

1. A register has been zeroed with either a SUB reg, reg instruction or an XOR reg, reg instruction,
2. A value is moved with sign extension into the same register's lower 16 bits; or a signed integer multiply is performed to the same register's lower 16 bits,
3. This register is then copied to an MMX™ technology register using the MOVD instruction prior to any other operations on the sign-extended value.

Specifically, the sign may be incorrectly extended into bits 16-31 of the MMX technology register. Only the MMX technology register is affected by this erratum.

The erratum only occurs when the 3 following steps occur in the order shown. The erratum may occur with up to 40 intervening instructions that do not modify the sign-extended value between steps 2 and 3.

1. XOR EAX, EAX
   or SUB EAX, EAX
2. MOVSX AX, BL
   or MOVSX AX, byte ptr <memory address> or MOVSX AX, BX
   or MOVSX AX, word ptr <memory address> or IMUL BL (AX implicit, opcode F6 /5)
   or IMUL byte ptr <memory address> (AX implicit, opcode F6 /5) or IMUL AX, BX (opcode 0F AF /r)
   or IMUL AX, word ptr <memory address> (opcode 0F AF /r) or IMUL AX, BX, 16 (opcode 6B /r ib)
   or IMUL AX, word ptr <memory address>, 16 (opcode 6B /r ib) or IMUL AX, 8 (opcode 6B /r ib)
   or IMUL AX, BX, 1024 (opcode 69 /r iw)
   or IMUL AX, word ptr <memory address>, 1024 (opcode 69 /r iw) or IMUL AX, 1024 (opcode 69 /r iw)
   or CBW
3. MOVD MM0, EAX

Note that the values for immediate byte/words are merely representative (i.e., 8, 16, 1024) and that any value in the range for the size may be affected. Also, note that this erratum may occur with "EAX" replaced with any 32-bit general purpose register, and "AX" with the corresponding 16-bit version of that replacement. "BL" or "BX" can be replaced with any 8-bit or 16-bit general purpose register. The CBW and IMUL (opcode F6 /5) instructions are specific to the EAX register only.

In the example, EAX is forced to contain 0 by the XOR or SUB instructions. Since the MOVSX, IMUL and CBW instructions listed should modify only bits 15:8 of EAX by sign extension, bits 31:16 of EAX should always contain 0. This implies that when MOVD copies EAX to MM0, bits 31:16 of MM0 should also be 0. Under certain

scenarios, bits 31:16 of MM0 are not 0, but are replicas of bit 15 (the 16th bit) of AX. This is noticeable when the value in AX after the MOVSX, IMUL or CBW instruction is negative, i.e., bit 15 of AX is a 1.

When AX is positive (bit 15 of AX is a 0), MOVD will always produce the correct answer. If AX is negative (bit 15 of AX is a 1), MOVD may produce the right answer or the wrong answer depending on the point in time when the MOVD instruction is executed in relation to the MOVSX, IMUL or CBW instruction.

IMPLICATION: The effect of incorrect execution will vary from unnoticeable, due to the code sequence discarding the incorrect bits, to an application failure. If the MMX technology-enabled application in which MOVD is used to manipulate pixels, it is possible for one or more pixels to exhibit the wrong color or position momentarily. It is also possible for a computational application that uses the MOVD instruction in the manner described above to produce incorrect data. Note that this data may cause an unexpected page fault or general protection fault.

WORKAROUND: There are two possible workarounds for this erratum:

1. Rather than using the MOVSX-MOVD or CBW-MOVD pairing to handle one variable at a time, use the sign extension capabilities (PSRAW, etc.) within MMX™ technology for operating on multiple variables. This would result in higher performance as well.
2. Insert another operation that modifies or copies the sign-extended value between the MOVSX/IMUL/CBW instruction and the MOVD instruction as in the example below:

```
XOR EAX, EAX (or SUB EAX, EAX)

MOVSX AX, BL (or other MOVSX, other IMUL or CBW instruction)

*MOV EAX, EAX

MOVD MM0, EAX
```

*Note: MOV EAX, EAX is used here as it is fairly generic. Again, EAX can be any 32-bit register.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

H24. Premature Execution of a Load Operation Prior to Exception Handler Invocation

PROBLEM: This erratum can occur with any of the following situations:

1. If an instruction that performs a memory load causes a code segment limit violation
2. If a waiting floating-point instruction or MMX™ instruction that performs a memory load has a floating-point exception pending, or
3. If an MMX instruction that performs a memory load and has either CR0.EM =1 (Emulation bit set), or a floating-point Top-of-Stack (FP TOS) not equal to 0, or a DNA exception pending.

If any of the above circumstances, occur it is possible that the load portion of the instruction will have executed before the exception handler is entered.

IMPLICATION: In normal code execution where the target of the load operation is to write back memory there is no impact from the load being prematurely executed, nor from the restart and subsequent re-execution of that instruction by the exception handler. If the target of the load is to uncached memory that has a system side-effect, restarting the instruction may cause unexpected system behavior due to the repetition of the side-effect.

WORKAROUND: Code which performs loads from memory that has side-effects can effectively workaround this behavior by using simple integer-based load instructions when accessing side-effect memory and by ensuring that all code is written such that a code segment limit violation cannot occur as a part of reading from side-effect memory.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

H25. Read Portion of RMW Instruction May Execute Twice

PROBLEM: When the Intel Mobile Celeron processor executes a read-modify-write (RMW) arithmetic instruction, with memory as the destination, it is possible for a page fault to occur during the execution of the store on the memory operand after the read operation has completed but before the write operation completes.

If the memory targeted for the instruction is UC (uncached), memory will observe the occurrence of the initial load before the page fault handler and again if the instruction is restarted.

IMPLICATION: This erratum has no effect if the memory targeted for the RMW instruction has no side-effects. If, however, the load targets a memory region that has side-effects, multiple occurrences of the initial load may lead to unpredictable system behavior.

WORKAROUND: Hardware and software developers who write device drivers for custom hardware that may have a side-effect style of design should use simple loads and simple stores to transfer data to and from the device. Then, the memory location will simply be read twice with no additional implications.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

H26. Intervening Writeback May Occur During Locked Transaction

PROBLEM: During a transaction which has the LOCK# signal asserted (e.g., a locked transaction), there is a potential for an explicit writeback caused by a previous transaction to complete while the bus is locked. The explicit writeback will only be issued by the processor which has locked the bus, and the lock signal will not be deasserted until the locked transaction completes, but the atomicity of a lock may be compromised by this erratum. Note that the explicit writeback is an expected cycle, and no memory ordering violations will occur. This erratum is, however, a violation of the bus lock protocol.

IMPLICATION: A chipset or third-party agent (TPA) which tracks bus transactions in such a way that locked transactions may only consist of a read-write or read-read-write-write locked sequence, with no transactions intervening, may lose synchronization of state due to the intervening explicit writeback. Systems using chipsets or TPAs which can accept the intervening transaction will not be affected.

WORKAROUND: The bus tracking logic of all devices on the system bus should allow for the occurrence of an intervening transaction during a locked transaction.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

H27. MC2_STATUS MSR Has Model-Specific Error Code and Machine Check Architecture Error Code Reversed

PROBLEM: The *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide,* documents that for the MC*i*_STATUS MSR, bits 15:0 contain the MCA (machine-check architecture) error code field and bits 31:16 contain the model-specific error code field. However, for the MC2_STATUS MSR, these bits have been reversed. For the MC2_STATUS MSR, bits 15:0 contain the model-specific error code field and bits 31:16 contain the MCA error code field.

IMPLICATION: A machine check error may be decoded incorrectly if this erratum on the MC2_STATUS MSR is not taken into account.

WORKAROUND: When decoding the MC2_STATUS MSR, reverse the two error fields.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

H28. Mixed Cacheability of Lock Variables Is Problematic in MP Systems

PROBLEM: This errata only affects multiprocessor systems where a lock variable address is marked cacheable in one processor and uncacheable in any others. The processors which have it marked uncacheable may stall indefinitely when accessing the lock variable. The stall is only encountered if:

- One processor has the lock variable cached, and is attempting to execute a cache lock.
- If the processor which has that address cached has it cached in its L2 only.
- Other processors, meanwhile, issue back to back accesses to that same address on the bus.

IMPLICATION: MP systems where all processors either use cache locks or consistent locks to uncacheable space will not encounter this problem. If, however, a lock variable's cacheability varies in different processors, and several processors are all attempting to perform the lock simultaneously, an indefinite stall may be experienced by the processors which have it marked uncacheable in locking the variable (if the conditions above are satisfied). Intel has only encountered this problem in focus testing with artificially generated external events. Intel has not currently identified any commercial software which exhibits this problem.

WORKAROUND: Follow a homogenous model for the memory type range registers (MTRRs), ensuring that all processors have the same cacheability attributes for each region of memory; do not use locks whose memory type is cacheable on one processor, and uncacheable on others. Avoid page table aliasing, which may produce a nonhomogenous memory model.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

H29. Thermal Sensor May Assert SMBALERT# Incorrectly

PROBLEM: The Intel Celeron Processor Mobile Module has a thermal sensor that monitors the processor core's temperature. Please note that desktop systems could have a similar thermal device. The thermal sensor asserts SMBALERT# if the processor temperature exceeds the temperature limits set in the Alarm Threshold Registers (T_{HIGH}, T_{LOW}). It also sets the corresponding Status Register bits to identify the cause of the interrupt. Figure 1 gives one example of the how the SMBALERT# signal could be used in a system.

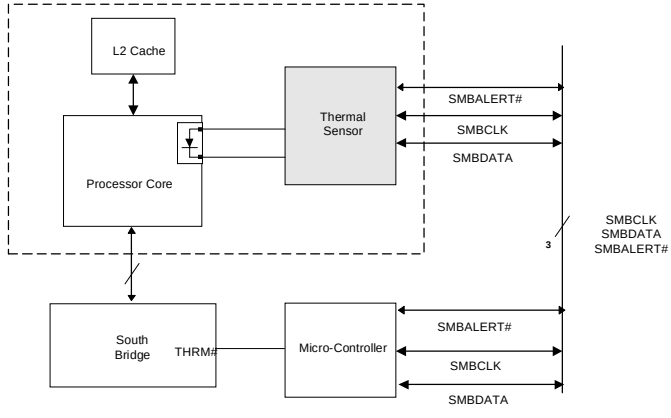


Figure 1. An Example of Microcontroller Driven Thermal Management

Under the conditions described below, the thermal sensor incorrectly generates the SMBALERT# interrupt. *All* of the following conditions must be met to trigger a false interrupt:

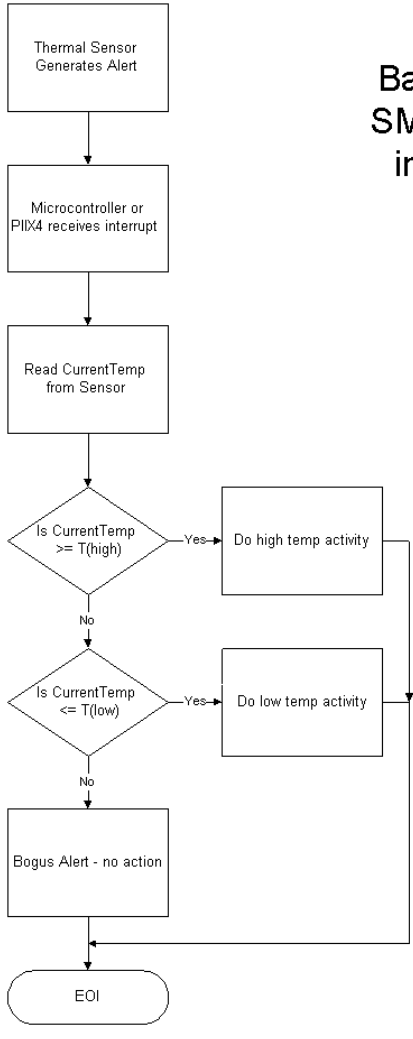
1. The thermal sensor must be in auto-convert mode.
2. The absolute value of the difference between the current temperature reading and the T_{HIGH} or T_{LOW} limit value must be less than or equal to 8 °C.
3. The current temperature reading must be different from the previous reading.

With a false assertion of SMBALERT#, the corresponding bit in the Status Register (L_{HIGH}, L_{LOW}, R_{HIGH}, and R_{LOW}) also will be incorrect.

IMPLICATION: There is no system impact from this erratum if temperature polling is used for processor thermal management. If the SMABLERT# interrupt is employed to manage processor thermal sensing, then servicing the false interrupt may result in premature system action depending on the software and hardware implementations used. The rate of the false interrupts is less than the auto-convert rate of the thermal sensor.

WORKAROUND: Three different (mutually exclusive) workarounds are possible:

1. Before servicing an interrupt from the thermal sensor, read and compare the processor thermal reading with the threshold limits (T_{HIGH} or T_{LOW}). Figures 2 and 3 provide basic flowcharts for the implementation of this workaround in an interrupt driven system.
2. If the firmware implemented polls the Status Register only, then before taking any action, re-read the temperature register and do a comparison with the alarm threshold limits (T_{HIGH} or T_{LOW}) to determine if the value is actually still within the temperature window.
3. Use a temperature polling scheme to monitor the processor temperature.



Basic flowchart for SMBus Alert driven implementations

Figure 2. Workaround Flowchart: SMBALERT#-Driven System

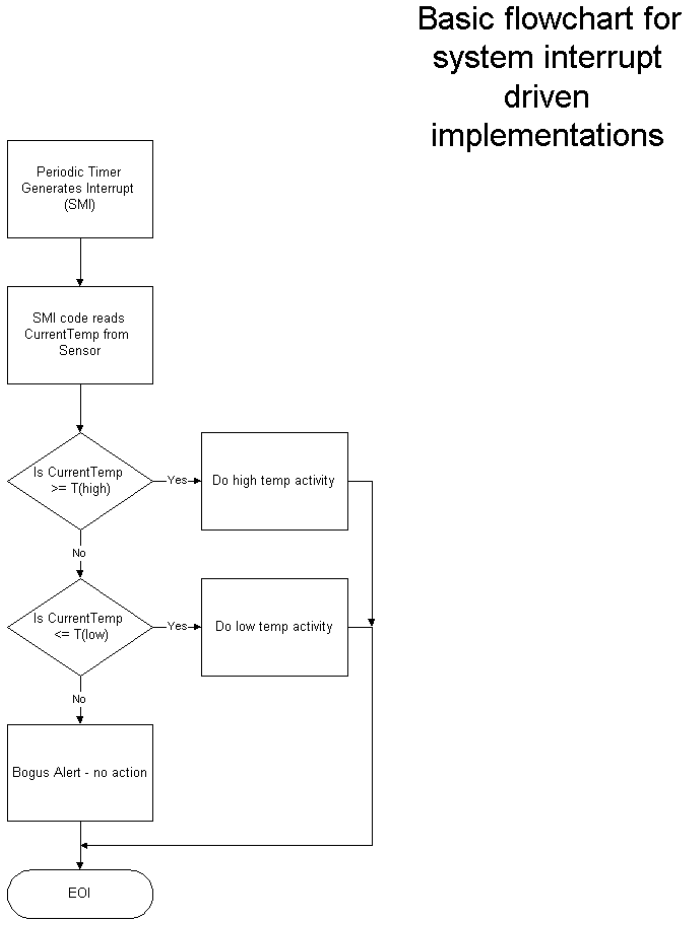


Figure 3. Workaround Flowchart: SMI#-Driven System

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

H30. MOV With Debug Register Causes Debug Exception

PROBLEM: When in V86 mode, if a MOV instruction is executed on debug registers, a general-protection exception (#GP) should be generated, as documented in the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*, Section 14.2. However, in the case when the general detect enable flag (GD) bit is set, the observed behavior is that a debug exception (#DB) is generated instead.

IMPLICATION: With debug-register protection enabled (e.g., the GD bit set), when attempting to execute a MOV on debug registers in V86 mode, a debug exception will be generated instead of the expected general-protection fault.

WORKAROUND: In general, operating systems do not set the GD bit when they are in V86 mode. The GD bit is generally set and used by debuggers. The debug exception handler should check that the exception did not occur in V86 mode before continuing. If the exception did occur in V86 mode, the exception may be directed to the general-protection exception handler.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

H31. Upper Four PAT Entries Not Usable With Mode B or Mode C Paging

PROBLEM: The Page Attribute Table (PAT) contains eight entries, which must all be initialized and considered when setting up memory types for the Intel Mobile Celeron processor. However, in Mode B or Mode C paging, the upper four entries do not function correctly for 4-Kbyte pages. Specifically, bit seven of page table entries that translate addresses to 4-Kbyte pages should be used as the upper bit of a three-bit index to determine the PAT entry that specifies the memory type for the page. When Mode B (CR4.PSE = 1) and/or Mode C (CR4.PAE) are enabled, the processor forces this bit to zero when determining the memory type regardless of the value in the page table entry. The upper four entries of the PAT function correctly for 2-Mbyte and 4-Mbyte large pages (specified by bit 12 of the page directory entry for those translations).

IMPLICATION: Only the lower four PAT entries are useful for 4KB translations when Mode B or C paging is used. In Mode A paging (4-Kbyte pages only), all eight entries may be used. All eight entries may be used for large pages in Mode B or C paging.

WORKAROUND: None identified.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

H32. Incorrect Memory Type May be Used When MTRRs Are Disabled

PROBLEM: If the Memory Type Range Registers (MTRRs) are disabled without setting the CR0.CD bit to disable caching, and the Page Attribute Table (PAT) entries are left in their default setting, which includes UC- memory type (PCD = 1, PWT = 0; see the *Addendum—Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide,* for details), data for entries set to UC- will be cached as if the memory type were writeback (WB). Also, if the page tables are set to a memory type other than UC-, then the effective memory type used will be that specified by the page tables and PAT. Any regions of memory normally forced to UC by the MTRRs (such as the VGA video region) may now be incorrectly cached and speculatively accessed.

Even if the CR0.CD bit is correctly set when the MTRRs are disabled and the PAT is left in its default state, then retries and out of order retirement of UC accesses may occur, contrary to the strong ordering expected for these transactions.

IMPLICATION: The occurrence of this erratum may result in the use of incorrect data and unpredictable processor behavior when running with the MTRRs disabled. Interaction between the mouse, cursor, and VGA video display leading to video corruption may occur as a symptom of this erratum as well.

WORKAROUND: Ensure that when the MTRRs are disabled, the CR0.CD bit is set to disable caching. This recommendation is described in *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*. If it is necessary to disable the MTRRs, first clear the PAT register before setting the CR0.CD bit, flushing the caches, and disabling the MTRRs to ensure that UC memory type is always returned and strong ordering is maintained.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

H33. Misprediction in Program Flow May Cause Unexpected Instruction Execution

PROBLEM: To optimize performance through dynamic execution technology, the P6 architecture has the ability to predict program flow. In the event of a misprediction, the processor will normally clear the incorrect prediction, adjust the EIP to the correct location, and flush out any instructions it may have fetched from the misprediction. In circumstances where a branch misprediction occurs, the correct target of the branch has already been opportunistically fetched into the streaming buffers, and the L2 cycle caused by the evicted cache line is retried by the L2 cache, the processor may fail to flush out the retirement unit before the speculative program flow is committed to a permanent state.

IMPLICATION: The results of this erratum may range from no effect to unpredictable application or OS failure. Manifestations of this failure may result in:

- Unexpected values in EIP,
- Faults or traps (e.g., page faults) on instructions that do not normally cause faults,
- Faults in the middle of instructions, or
- Unexplained values in registers/memory at the correct EIP.

WORKAROUND: It is possible for BIOS code to contain a workaround for this erratum.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

H34. Data Breakpoint Exception in a Displacement Relative Near Call May Corrupt EIP

PROBLEM: If a data breakpoint is programmed at the memory location where the stack push of a near call is performed, the processor will update the stack and ESP appropriately, but may skip the code at the destination of the call. Hence, program execution will continue with the next instruction immediately following the call, instead of the target of the call.

IMPLICATION: The failure mechanism for this erratum is that the call would not be taken; therefore, instructions in the called subroutine would not be executed. As a result, any code relying on the execution of the subroutine will behave unpredictably.

WORKAROUND: Do not program a data breakpoint exception on the stack where the push for the near call is performed.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

H35. System Bus ECC Not Functional With 2:1 Ratio

PROBLEM: If a processor is underclocked at a core frequency to system bus frequency ratio of 2:1 and system bus ECC is enabled, the system bus ECC detection and correction will negatively affect internal timing dependencies.

IMPLICATION: If system bus ECC is enabled, and the processor is underclocked at a 2:1 ratio, the system may behave unpredictably due to these timing dependencies.

WORKAROUND: All bus agents that support system bus ECC must disable it when a 2:1 ratio is used.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

H36. Fault on REP CMPS/SCAS Operation May Cause Incorrect EIP

PROBLEM: If either a General Protection Fault, Alignment Check Fault or Machine Check Exception occur during the first iteration of a REP CMPS or a REP SCAS instruction, an incorrect EIP may be pushed onto the stack of the event handler if all the following conditions are true:

- The event occurs on the initial load performed by the instruction(s),
- The condition of the zero flag before the repeat instruction happens to be opposite of the repeat condition (e.g., REP/REPE/REPZ CMPS/SCAS with ZF = 0 or RENE/REPNZ CMPS/SCAS with ZF = 1), and
- The faulting micro-op and a particular micro-op of the REP instruction are retired in the retirement unit in a specific sequence.

The EIP will point to the instruction following the REP CMPS/SCAS instead of pointing to the faulting instruction.

IMPLICATION: The result of the incorrect EIP may range from no effect to unexpected application/OS behavior.

WORKAROUND: None identified at this time.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

H37. RDMSR or WRMSR To Invalid MSR Address May Not Cause GP Fault

PROBLEM: The RDMSR and WRMSR instructions allow reading or writing of MSRs (Model Specific Registers) based on the index number placed in ECX. The processor should reject access to any reserved or unimplemented MSRs by generating #GP(0). However, there are some invalid MSR addresses for which the processor will not generate #GP(0).

IMPLICATION: For RDMSR, undefined values will be read into EDX:EAX. For WRMSR, undefined processor behavior may result.

WORKAROUND: Do not use invalid MSR addresses with RDMSR or WRMSR.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

H38. SYSENTER/SYSEXIT Instructions Can Implicitly Load "Null Segment Selector" to SS and CS Registers

PROBLEM: According to the processor specification, attempting to load a null segment selector into the CS and SS segment registers should generate a General Protection Fault (#GP). Although loading a null segment selector to the other segment registers is allowed, the processor will generate an exception when the segment register holding a null selector is used to access memory.

However, the SYSENTER instruction can implicitly load a null value to the SS segment selector. This can occur if the value in SYSENTER_CS_MSR is between FFF8h and FFFBh when the SYSENTER instruction is executed. This behavior is part of the SYSENTER/SYSEXIT instruction definition; the content of the SYSTEM_CS_MSR is always incremented by 8 before it is loaded into the SS. This operation will set the null bit in the segment selector if a null result is generated, but it does not generate a #GP on the SYSENTER instruction itself. An exception will be generated as expected when the SS register is used to access memory, however.

The SYSEXIT instruction will also exhibit this behavior for both CS and SS when executed with the value in SYSENTER_CS_MSR between FFF0h and FFF3h, or between FFE8h and FFEBh, inclusive.

IMPLICATION: These instructions are intended for operating system use. If this erratum occurs (and the OS does not ensure that the processor never has a null segment selector in the SS or CS segment registers), the processor's behavior may become unpredictable, possibly resulting in system failure.

WORKAROUND: Do not initialize the SYSTEM_CS_MSR with the values between FFF8h and FFFBh, FFF0h and FFF3h, or FFE8h and FFEBh before executing SYSENTER or SYSEXIT.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

H39. PRELOAD Followed by EXTEST Does Not Load Boundary Scan Data

PROBLEM: According to the IEEE 1149.1 Standard, the EXTEST instruction would use data "typically loaded onto the latched parallel outputs of boundary-scan shift-register stages using the SAMPLE/PRELOAD instruction prior to the selection of the EXTEST instruction." As a result of this erratum, this method cannot be used to load the data onto the outputs.

IMPLICATION: Using the PRELOAD instruction prior to the EXTEST instruction will not produce expected data after the completion of EXTEST.

WORKAROUND: None identified at this time.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

H40. Far Jump to New TSS With D-bit Cleared May Cause System Hang

PROBLEM: A task switch may be performed by executing a far jump through a task gate or to a new Task State Segment (TSS) directly. Normally, when such a jump to a new TSS occurs, the D-bit (which indicates that the page referenced by a Page Table Entry (PTE) has been modified) for the PTE which maps the location of the previous TSS will already be set and the processor will operate as expected. However, if the D-bit is clear at the time of the jump to the new TSS, the processor will hang.

IMPLICATION: If an OS is used which can clear the D-bit for system pages, and which jumps to a new TSS on a task switch, then a condition may occur which results in a system hang. Intel has not identified any commercial software which may encounter this condition; this erratum was discovered in a focused testing environment.

WORKAROUND: Ensure that OS code does not clear the D-bit for system pages (including any pages that contain a task gate or TSS). Use task gates rather than jumping to a new TSS when performing a task switch.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

H41. Incorrect Chunk Ordering May Prevent Execution of the Machine Check Exception Handler After BINIT#

PROBLEM: If a catastrophic bus error is detected which results in a BINIT# assertion, and the BINIT# assertion is propagated to the processor's L2 cache at the same time that data is being sent to the processor, then the data may become corrupted in the processor's L1 cache.

IMPLICATION: Since BINIT# assertion is due to a catastrophic event on the bus, the corrupted data will not be used. However, it may prevent the processor from executing the Machine Check Exception (MCE) handler, causing the system to hang.

WORKAROUND: None identified at this time.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

H42. Resume Flag May Not Be Cleared After Debug Exception

PROBLEM: The Resume Flag (RF) is normally cleared by the processor after executing an instruction which causes a debug exception (#DB). In the process of determining whether the RF needs to be cleared after executing the instruction, the processor uses an internal register containing stale data. The stale data may unpredictably prevent the processor from clearing the RF.

IMPLICATION: If this erratum occurs, further debug exceptions will be disabled.

WORKAROUND: None identified at this time.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

DOCUMENTATION CHANGES

The Documentation Changes listed in this section apply to the *Intel Architecture Software Developer's Manual*, *Volumes 1, 2, and 3*. All Documentation Changes will be incorporated into a future version of the appropriate Intel Mobile Celeron processor documentation.

NOTE

The *Intel Architecture Software Developer's Manual*, *Volumes 1, 2, and 3* applies to all P6 family processors, and therefore some of the Documentation Changes in this section may not pertain to the Intel Mobile Celeron processor specifically.

NOTE

Most of the Documentation Changes previously listed in this section have all been incorporated into an updated version of the *Intel Architecture Software Developer's Manual, Volumes 1, 2, and 3* (Order Numbers 243190-002, 243191-002, and 243192-002, respectively). The updated versions can be ordered by contacting the Intel Literature Center.

H1. SMBus Data Setup Time

The SMBus data signal has a min setup time (min Tsu: DAT) of 250ns, as published in the *System Management Bus Specification*, Rev 1.0. This parameter is currently specified to 800 ns for the Mobile Pentium® II processor (mini-cartridge) and Intel Pentium II Processor Mobile Module products (see table below).

SMBus Data Min Setup

Parameter	Description	Spec	Change to
min Tsu: DAT	min SMBus Data Setup Time	250 ns	800 ns

SPECIFICATION CLARIFICATIONS

The Specification Clarifications listed in this section apply to the *Intel® Mobile Celeron™ Processor (BGA)* datasheet (Order Number#245106-001), or the *Intel® Mobile Celeron Processor in Mobile Module MMC-1* (Order Number #245101-001), or the *Intel® Mobile Celeron Processor in Mobile Module MMC-2* (Order Number #245102-001). All Specification Changes will be incorporated into a furture version on the appropriate Intel Celeron processor documentation.

NOTE

The *Intel Architecture Software Developer's Manual*, *Volumes 1, 2, and 3,* applies to all P6 family processors, and therefore some of the specification clarifications in this section may not pertain to the Intel Mobile Celeron processor or the Intel Celeron Processor Mobile Module specifically.

NOTE

The Specification Clarifications previously listed in this section have all been incorporated into an updated version of the *Intel Architecture Software Developer's Manual, Volumes 1, 2, and 3* (Order Numbers 243190-002, 243191-002, and 243192-002, respectively). The updated versions can be ordered by contacting the Intel Literature Center.

SPECIFICATION CHANGES

The Specification Changes listed in this section apply to the *Intel® Mobile Celeron™ Processor (BGA)* datasheet (Order Number#245106-001), or the *Intel® Mobile Celeron Processor in Mobile Module MMC-1* (Order Number #245101-001), or the *Intel® Mobile Celeron Processor in Mobile Module MMC-2* (Order Number #245102-001). All Specification Changes will be incorporated into a future version of the appropriate Intel Celeron processor documentation.

NOTE

The Specification Changes previously listed in this section have all been incorporated into an updated version of the *Intel Architecture Software Developer's Manual, Volumes 1, 2, and 3* (Order Numbers 243190-002, 243191-002, and 243192-002, respectively). The updated versions can be ordered by contacting the Intel Literature Center.